Schedulability-Driven Partitioning and Mapping for Multi-Cluster Real-Time Systems

Pop, Paul; Eles, Petru; Peng, Zebo

Link to article, DOI:
10.1109/EMRTS.2004.1311010

Publication date:
2004

Document Version
Publisher's PDF, also known as Version of record

Schedulability-Driven Partitioning and Mapping for Multi-Cluster Real-Time Systems

Paul Pop, Petru Eles, Zebo Peng, Viacheslav Izosimov
Dept. of Computer and Information Science
Linköping University, SE-581 83, Linköping, Sweden
{paupo, petel, zebpe, viaiz}@ida.liu.se

Abstract

We present an approach to partitioning and mapping for multi-cluster embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. We have proposed a schedulability analysis for such systems, including a worst-case queuing delay analysis for the gateways responsible for routing inter-cluster traffic. Based on this analysis, we address design problems characteristic to multi-clusters: partitioning of the system functionality into time-triggered and event-triggered domains, and mapping of processes onto architecture nodes. We present a branch-and-bound algorithm for solving these problems. Our algorithm is able to find schedulable implementations under limited resources, achieving an efficient utilization of the system. The developed algorithms are evaluated using extensive experiments and a real-life example.

1. Introduction

Process scheduling and schedulability analysis have been intensively studied in the past decades. The reader is referred to [2] for a survey on these topics. Work in the area of scheduling and schedulability analysis diversified significantly by considering particular communication protocols, such as Token-Ring [12], Controller Area Network (CAN) [1, 5, 20], ATM [7], and time-division multiple access (TDMA) [21] protocols.

An increasing number of real-time applications are today implemented using distributed architectures consisting of interconnected *clusters* of processors. Each such cluster has its own communication protocol and two clusters communicate via a *gateway,* a node connected to both of them. This type of architectures is used in several application areas: vehicles, factory systems, networks on chip, etc.

Considering, for example, the automotive industry, the way functionality has been distributed on an architecture has evolved over time. Initially, each function was implemented on a dedicated hardware component. However, in order to use the resources more efficiently and reduce costs, several functions have later been integrated in one node and, at the same time, certain functionality has been distributed over several nodes. Although an application is typically distributed over one single cluster, we begin to see applications that are distributed across several clusters. This trend is driven by the need to further reduce costs, improve resource usage, but also by application constraints like having to be physically close to particular sensors and actuators. Moreover, not only are these applications distributed across networks, but their functions can exchange critical information through the gateway nodes. Such applications are inherently difficult to analyze and design.

There are two basic approaches for handling tasks in real-time applications [10]. In the *event-triggered* (ET) approach, activities are initiated whenever a particular event is noted. In the *time-triggered* (TT) approach, activities are initiated at predetermined points in time. There has been a long debate in the real-time and embedded systems communities concerning the advantages of TT and ET approaches [3, 10, 22]. An interesting comparison, from a more industrial, in particular automotive, perspective, can be found in [11]. The conclusion there is that one has to choose the right approach depending on the particularities of the processes. This means not only that there is no single "best" approach to be used, but also that inside a certain application the two approaches can be used together, some processes being TT and others ET.

In [13] we have addressed design problems for systems where the TT and ET activities *share the same* processor and bus. A fundamentally different architectural approach to heterogeneous TT/ET systems is that of heterogeneous multi-clusters, where each cluster can be either TT or ET:

- In a *time-triggered cluster* (TTC) processes and messages are scheduled according to a static cyclic policy, with the bus implementing a TDMA protocol such as, the time-triggered protocol (TTP) [10].
- On *event-triggered clusters* (ETC) the processes are scheduled according to a priority based preemptive approach, while messages are transmitted using the priority-based CAN bus [4].

In this context, in [16] we have proposed an approach to schedulability analysis for multi-cluster distributed embedded systems. Starting from such an analysis, in this paper, we address specific design issues for multi-cluster systems: partitioning an application between the TT and ET clusters, and mapping the functionality of the application on the heterogeneous nodes of a cluster, such that the timing constraints of the final implementation are guaranteed. Our design space exploration approach is based on an efficient branch-and-bound algorithm.

The paper is organized in seven sections. The next section presents the application model as well as the hardware and software architecture of our systems. Section 3 presents the partitioning and mapping problem we are addressing in this paper, and Section 4 presents the schedulability analysis for multi-clusters and our proposed partitioning and mapping branch-and-bound strategy is presented in Section 5. The last two sections present the experimental results and conclusions.

2. System Architecture and Application Model

2.1 Application Model

We model an application Γ as a set of directed, acyclic, polar graphs $G_i(V, E) \in \Gamma$ (see Figure 1). Each node $P_i \in V$ represents

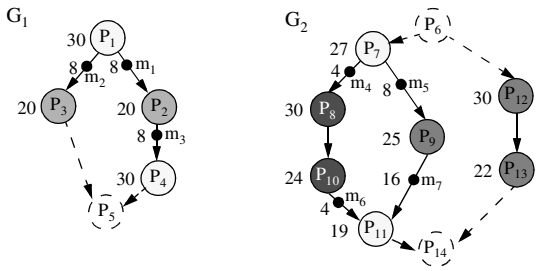

Figure 1. An Application Model Example

one process. An edge $e_{ij} \in E$ from P_i to P_j indicates that this output of P_i is an input to P_j. A process can be activated after all its inputs have arrived and it issues its outputs when it terminates. The communication time between processes mapped on the same processor is considered to be part of the process worst-case execution time and is not modeled explicitly. Communication between processes mapped to different processors is performed by message passing over the buses and, if needed, through the gateway. Such message passing is modeled as a communication process inserted on the arc connecting the sender and the receiver process (depicted with black dots in Figure 1).

The mapping of a process graph $G(V, E)$ is given by a function $\mathcal{M}: V \rightarrow \mathcal{N}$, where $\mathcal{N}$ is the set of programmable processors in the architecture. For a process $P_i \in V$, $\mathcal{M}(P_i)$ is the node to which P_i is assigned for execution. Each process P_i can potentially be mapped on several nodes. Let $\mathcal{N}_{P_i} \subseteq \mathcal{N}$ be the set of nodes to which P_i can potentially be mapped. We consider that for each $N_k \in \mathcal{N}_{P_i}$, we know the worst-case execution time $C_{P_i}^{N_k}$ of process P_i, when executed on N_k. We also consider that the size of the messages is given. Processes and messages activated based on events also have a uniquely assigned priority, p_{P_i} for processes and p_{m_i} for messages.

All processes and messages belonging to a process graph G_i have the same period $T_i = T_{G_i}$ which is the period of the process graph. A deadline $D_{G_i} \leq T_{G_i}$ is imposed on each process graph G_i. In addition, processes can have associated individual release times and deadlines. If communicating processes are of different periods, they are combined into a hyper-graph capturing all process activations for the hyper-period (LCM of all periods).

2.2 Hardware Architecture

We consider architectures consisting of two interconnected clusters. A *cluster* is composed of nodes which share a broadcast communication channel. Let $\mathcal{N}_T$ ($\mathcal{N}_E$) be the set of nodes on the TTC (ETC). Every *node* $N_i \in \mathcal{N}_T \cup \mathcal{N}_E$ consists, among others, of a communication controller and a CPU. The gateway node N_G is connected to both types of clusters, and has two communication controllers, for TTP and CAN, respectively. The communication controllers implement the protocol services and run independently of the node's CPU. Communication with the CPU is performed through a *message base interface* (MBI), see Figure 2.

Communication between the nodes on a TTC is based on the TTP [10]. The bus access scheme is TDMA, where each node N_i, including the gateway node, can transmit only during a predetermined time interval, the so called TDMA slot S_i. In such a slot, a node can send several messages packed in a frame. A sequence of slots corresponding to all the nodes in the TTC is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table for the TTP controller which has to know when to send/receive a frame to/from the communication channel.

On an ETC the CAN [4] protocol is used for communication. The CAN bus is a priority bus that employs a collision avoidance mechanism, whereby the node that transmits the message with the highest priority wins the contention. Message priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

The approaches presented in this paper can be easily extended to cluster configurations where there are several ETCs and TTCs interconnected by gateways.

2.3 Software Architecture

A real-time kernel is responsible for the activation of processes and transmission of messages on each node. On a TTC, the processes are activated based on the local schedule tables, and messages are transmitted according to the MEDL. On an ETC, we have a scheduler that decides on the activation of ready processes and transmission of messages, based on their priorities.

In Figure 2 we illustrate our message passing mechanism. Here we concentrate on the communication between processes located on different clusters. For message passing details within a TTC the reader is directed to [15], while the infrastructure needed for communications on an ETC has been detailed in [20].

Let us consider the example in Figure 2, where we have the process graph G_1 from Figure 1 mapped on the two clusters. Processes P_1 and P_4 are mapped on node N_1 of the TTC, while P_2 and P_3 are mapped on node N_2 of the ETC. P_1 sends messages m_1 and m_2 to processes P_2 and P_3, respectively, while P_2 sends m_3 to P_4.

The transmission of messages from the TTC to the ETC takes place in the following way (see Figure 2). P_1, which is statically scheduled, is activated according to the schedule table, and when it finishes it calls the send kernel function in order to send m_1 and m_2, indicated in the figure by number (1). Messages m_1 and m_2 have to be sent from node N_1 to node N_2. At a certain time, known from the schedule table, the kernel transfers m_1 and m_2 to the TTP controller by packing them into a frame in the MBI. Later on, the TTP controller knows from its MEDL when it has to take the frame from the MBI, in order to broadcast it on the bus. In our example, the timing information in the schedule table of the kernel and the MEDL is determined in such a way that the broadcasting of the

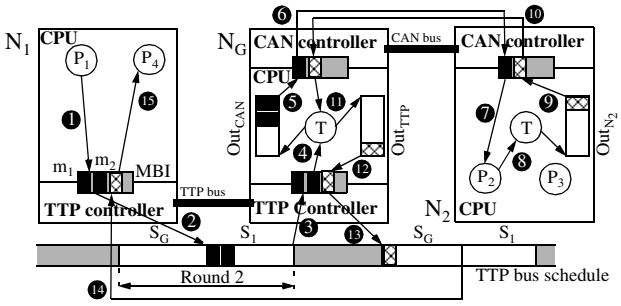

Figure 2. A Message Passing Example

frame is done in the slot S_1 of round two (2). The TTP controller of the gateway node N_G knows from its MEDL that it has to read a frame from slot S_1 of round two and to transfer it into its MBI (3). Invoked periodically, having the highest priority on node N_G, and with a period which guarantees that no messages are lost, the gateway process T copies messages m_1 and m_2 from the MBI to the TTP-to-CAN priority-ordered message queue Out_{CAN} (4). The highest priority message in the queue, in our case m_1, will tentatively be broadcast on the CAN bus (5). Whenever message m_1 will be the highest priority message on the CAN bus, it will successfully be broadcast and will be received by the interested nodes, in our case node N_2 (6). The CAN communication controller of node N_2 receiving m_1 will copy it in the transfer buffer between the controller and the CPU, and raise an interrupt which will activate a delivery process, responsible to activate the corresponding receiving process, in our case P_2, and hand over message m_1 that finally arrives at the destination (7).

Message m_3 (depicted in Figure 2 as a hashed rectangle) sent by process P_2 from the ETC will be transmitted to process P_4 on the TTC. The transmission starts when P_2 calls its send function and the delivery process D, which manages the priority-ordered Out_{N_2} queue, enqueues m_3 in Out_{N_2} (8). When m_3 has the highest priority on the bus, it will be removed from the queue (9) and will be broadcast on the CAN bus (10), arriving at the gateway's CAN controller where it raises an interrupt. Based on this interrupt, the gateway transfer process T is activated, and m_3 is placed in the Out_{TTP} FIFO queue (11). The gateway node N_G is only able to broadcast on the TTC in the slot S_G of the TDMA rounds circulating on the TTP bus. According to the MEDL of the gateway, a set of messages not exceeding $size_{S_G}$ of the slot S_G will be removed from the front of the Out_{TTP} queue in every round, and will be packed in the S_G slot (12). Once the frame is broadcast (13) it will arrive at node N_1 (14), where all the messages in the frame will be copied in the input buffers of the destination processes (15). Process P_4 is activated according to the schedule table, which has to be constructed such that it accounts for the worst-case communication delay of message m_3, bounded by the analysis in Section 4.1 and thus when P_4 starts executing it will find m_3 in its input buffer.

As part of our schedulability-driven partitioning and mapping approach, we determine a partitioning and mapping of processes to the nodes of the architecture, generate all the local schedule tables and MEDLs on the TTC and the message and process priorities for the activities on the ETC, such that the global system is schedulable.

3. Multi-Cluster Partitioning and Mapping

Considering the type of applications and systems described in Section 2, and using the multi-cluster scheduling approach outlined in the next section, several design optimization problems can be addressed. In this paper, we address problems which are characteristic to applications distributed across multi-cluster systems consisting of heterogeneous TT and ET networks. In particular, we are interested in the following issues:

1. partitioning of the processes of an application into time-triggered and event-triggered domains, and their mapping to the nodes of the clusters;
2. scheduling of processes and messages;

The goal is to produce an implementation which meets all the timing constraints of the application.

In this paper, by partitioning we denote the decision whether a certain process should be assigned to the TT or the ET domain (and, implicitly, to a TTC or an ETC, respectively). Mapping a process means assigning it to a particular node inside a cluster.

Very often, the partitioning decision is taken based on the experience and preferences of the designer, considering aspects like the functionality implemented by the process, the hardness of the constraints, sensitivity to jitter, legacy constraints, etc. Let $\mathcal{P}$ be the set of processes in the application Γ. We denote with $\mathcal{P}_T \subseteq \mathcal{P}$ the subset of processes which the designer has assigned to the TT cluster, while $\mathcal{P}_E \subseteq \mathcal{P}$ contains processes which are assigned to the ET cluster.

Many processes, however, do not exhibit certain particular features or requirements which obviously lead to their implementation as TT or ET activities. The subset $\mathcal{P}^+ = \mathcal{P} \setminus (\mathcal{P}_T \cup \mathcal{P}_E)$ of processes could be assigned to any of the TT or ET domains. Decisions concerning the partitioning of this set of activities can lead to various trade-offs concerning, for example, the schedulability properties of the system, the amount of communication exchanged through the gateway, the size of the schedule tables, etc.

For part of the partitioned processes, the designer might have already decided their mapping. For example, certain processes, due to constraints like having to be close to sensors/actuators, have to be physically located in a particular hardware unit. They represent the sets $\mathcal{P}_T^M \subseteq \mathcal{P}_T$ and $\mathcal{P}_E^M \subseteq \mathcal{P}_E$ of already mapped TT and ET processes, respectively. Consequently, we denote with $\mathcal{P}_T^* = \mathcal{P}_T \setminus \mathcal{P}_T^M$ the TT processes for which the mapping has not yet been decided, and similarly, with $\mathcal{P}_E^* = \mathcal{P}_E \setminus \mathcal{P}_E^M$ the unmapped ET processes. The set $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$ then represents all the unmapped processes in the application.

The mapping of messages is decided implicitly by the mapping of processes. Thus, a message exchanged between two processes on the TTC (ETC) will be mapped on the TTP bus (CAN bus) if these processes are allocated to different nodes. If the communication takes place between two clusters, two message instances will be created, one mapped on the TTP bus and one on the CAN bus. The first message is sent from the sender node to the gateway, while the second message is sent from the gateway to the receiving node.

Let us illustrate some of the issues related to partitioning in such a context. In the example presented in Figure 3 we have an application[1] with six processes, P_1 to P_6, and four nodes, N_1 and N_2 on the TTC, N_3 on the ETC and the gateway node N_G. The worst-case execution times on each node are given to the right of the applica-

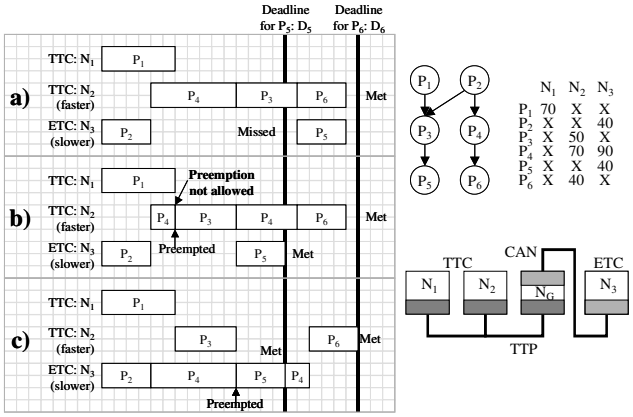

Figure 3. Partitioning Example

1. Communications are ignored for this example.

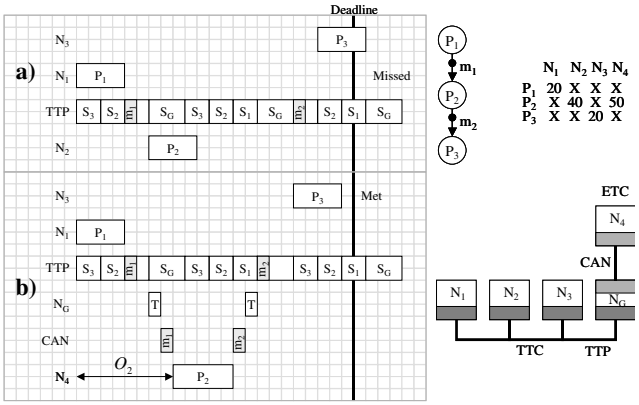

Figure 4. Mapping Example

tion graph. Note that N_2 is faster than N_3, and an "X" in the table means that the process is not allowed to be mapped on that node. The mapping of P_1 is fixed on N_1, P_3 and P_6 are mapped on N_2, P_2 and P_5 are fixed on N_3, and we have to decide how to partition P_4 between the TT and ET domains. Let us also assume that process P_5 is the highest priority process on N_3. In addition, P_5 and P_6 have each a deadline, D_5 and D_6, respectively, as illustrated in the figure by thick vertical lines.

We can observe that although P_3 and P_4 do not have individual deadlines, their mapping and scheduling have a strong impact on their successors, P_5 and P_6, respectively, which are deadline constrained. Thus, we would like to map P_4 such that not only P_3 can start on time, but P_4 also starts soon enough to allow P_6 to meet its deadline.

As we can see from Figure 3a, this is impossible to achieve by mapping P_4 on the TTC node N_2. It is interesting to observe that, if preemption would be allowed in the TT domain, as in Figure 3b, both deadlines could be met. This, however, is impossible on the TTC where preemption is not allowed. Both deadlines can be met only if P_4 is mapped on the slower ETC node N_3, as depicted in Figure 3c. In this case, although P_4 competes for the processor with P_5, due to the preemption of P_4 by the higher priority P_5, all deadlines are satisfied.

It is to be noted that, in principle, an effect similar to preemption can be achieved on the TTC, by deciding offline the preemption points, e.g., dividing process P_4 in two parts. However, deciding offline the preemption points on the TTC such that the application is schedulable, is a computationally very complex procedure. On the ETC, however, the preemption is handled automatically by the scheduler.

For a multi-cluster architecture the communication infrastructure has an important impact on the design and, in particular, the mapping decisions. Let us consider the example in Figure 4. We assume that P_1 is mapped on node N_1 and P_3 on node N_3 on the TTC, and we are interested to map process P_2. P_2 is allowed to be mapped on the TTC node N_2 or on the ETC node N_4, and its execution times are depicted in the table to the right of the application graph.

In order to meet the deadline, one would map P_2 on the node it executes fastest, N_2 on the TTC, see Figure 4a. However, this will lead to a deadline miss due to the TTP slot configuration which introduces communication delays. The application will meet the deadline only if P_2 is mapped on the slower node, i.e., node N_4 in the case in Figure 4b[1]. Not only is N_4 slower than N_2, but mapping P_2 on N_4 will place P_2 on a different cluster than P_1 and P_3, introducing extra communication delays through the gateway node. However, due to the actual communication configuration, the mapping alternative in Figure 4b is desirable.

4. Multi-Cluster Scheduling

Once a partitioning and a mapping is decided, the processes and messages have to be scheduled. For the TTC this means building the schedule tables, while for the ETC the priorities of the ET processes and messages have to be determined and their schedulability analyzed. The aim is to find out if a system is schedulable, i.e. all the timing constraints are met.

For the ETC we use a *response time analysis* technique, where the schedulability test consists of the comparison between the worst-case response time r_i of a process P_i and its deadline D_i. Response time analysis of data dependent processes with static priority preemptive scheduling has been proposed in [18, 19], and has been also extended to consider the CAN protocol [20]. The authors use the concept of *offset* in order to handle data dependencies. Thus, each process P_i is characterized by an offset O_i, measured from the start of the process graph, that indicates the earliest possible start time of P_i.

Let us consider the example in Figure 4. In the system configuration considered in Figure 4b we consider that, on the TTP bus, node N_1 transmits in the third slot (S_1) of the TDMA round. In this setting, the offset O_2 is determined as depicted in Figure 4b, since process P_2 cannot start before receiving m_1. The same is true for messages, their offset indicating the earliest possible transmission time.

Determining the schedulability of an application mapped on a multi-cluster system cannot be addressed separately for each type of cluster, since the inter-cluster communication creates a circular dependency: the static schedules determined for the TTC influence through the offsets the worst-case response times of the processes on the ETC, which on their turn influence the schedule table construction on the TTC.

In our response time analysis we consider the influence between the two clusters by making the following observations:

- The start time of process P_i in a schedule table on the TTC is its offset O_i.
- The worst-case response time r_i of a TT process is its worst-case execution time, i.e. $r_i = C_i$ (TT processes are not preemptable).
- The worst-case response times of the messages exchanged between two clusters have to be calculated according to the schedulability analysis described in Section 4.1.
- The offsets have to be set by a scheduling algorithm such that the precedence relationships are preserved. This means that, if process P_B depends on process P_A, the following condition must hold: $O_B \geq O_A + r_A$. Note that for the processes on a TTC receiving messages from the ETC this translates to setting the start times of the processes such that a process is not activated before the worst-case arrival time of the message from the ETC. In general, offsets on the TTC are set such that all the necessary messages are present at the process invocation.

MultiClusterScheduling in Figure 5 receives as input:

1. Process T in Figure 4b executing on the gateway node N_G is responsible for transferring messages between the TTP and CAN controllers.

```
MultiClusterScheduling(Γ, 𝓜, β, π)
 1 -- assign initial values to offsets
 2 for each Oi Œ ϕ do
 3       Oi = initial value
 4 end for
 5
 6 -- iteratively improve the offsets and response times
 7 repeat
 8       -- determine the response times based on
 9       -- the current values for the offsets
10       ρ = ResponseTimeAnalysis(Γ, 𝓜, β, π, ϕ)
11       -- determine the offsets based on
12       -- the current values for the response times
13       ϕ = StaticScheduling(Γ, 𝓜, β, ρ)
14 until ϕ not changed
15
16 return ϕ, ρ
end MultiClusterScheduling
```

Figure 5. The MultiClusterScheduling Algorithm

- the application Γ and its mapping $\mathcal{M}$;
- the TTC bus configuration β, indicating the sequence and size of the slots in a TDMA round on the TTC;
- the priorities of the processes and messages on the ETC, captured by π;

and produces as output:

- the set ϕ of the offsets corresponding to each process and message in the system (on the TTC the offsets practically represent the local schedule tables and MEDLs);
- the worst-case response times ρ for the processes and messages on the ETC cluster.

The algorithm starts by assigning to all offsets an initial value obtained by a static scheduling algorithm applied on the TTC without considering the influence from the ETC (lines 2–4). The worst-case response times of all processes and messages in the ETC are then calculated according to the analysis in Section 4.1 by using the ResponseTimeAnalysis function (line 10). Based on the worst-case response times, offsets of the TT processes can be defined such that all messages received from the ETC cluster are present at process invocation. Considering these offsets as constraints, a static scheduling algorithm can derive the schedule tables and MEDLs of the TTC cluster (line 13). For this purpose we use a list scheduling based approach presented in [6]. Once new values have been determined for the offsets, they are fed back to the response time calculation function, thus obtaining new and *tighter* (i.e., smaller, less pessimistic) values for the worst-case response times. The algorithm stops when the response times cannot be further tightened and, consequently, the offsets remain unchanged. Termination is guaranteed if processor and bus loads are smaller than 100% (see Section 4.1) and deadlines are smaller than the periods.

4.1 Schedulability Analysis

The analysis in this section is used in the ResponseTimeAnalysis function in order to determine the worst-case response times for processes and messages on the ETC. It receives as input the application Γ and its mapping $\mathcal{M}$, the offsets ϕ and the ETC priorities π, and it produces the set ρ of worst-case response times. For the response time calculation, we have extended the framework provided by [19, 20] in order to support the communication infrastructure of a multi-cluster system. Thus, the worst-case response time of a process P_i on the ETC is:

$$r_i = \max_{q = 0, 1, 2\ldots} \left(\max_{\forall P_j \in G} \left(w_i(q) + O_j + J_j - T_G \left(q + \left\lceil \frac{O_j + J_j - O_i - J_i}{T_G} \right\rceil \right) - O_i \right) \right) \quad (1)$$

where T_G the period of the process graph G, O_i and O_j are offsets of processes P_i and P_j, respectively, and J_i and J_j are the jitters of P_i and P_j. The jitter is the worst-case delay between the arrival of a process and its release. In Equation (1), q is the number of busy periods being examined, and $w_i(q)$ is the width of the level-i busy period starting at time qT_G [19]:

$$w_i(q) = B_i + (q+1)C_i + I_i \,. \quad (2)$$

In the previous equation, the blocking term B_i represents interference from lower priority processes that are in their critical section and cannot be interrupted, and C_i represents the worst-case execution time of process P_i. The last term captures the interference I_i from higher priority processes. The reader is directed to [19] for the details of the interference calculation.

Although this analysis is exact (both necessary and sufficient), it is computationally infeasible to evaluate. Hence, [19] proposes a feasible but not exact analysis (sufficient but not necessary) for solving Equation (1). Our MultiClusterScheduling algorithm uses the feasible analysis provided in [19] for deriving the worst-case response time of a process P_i.

Regarding the worst-case response time of messages, we have extended the analysis from [20] and applied it for frames on the CAN bus:

$$r_m = \max_{q = 0, 1, 2\ldots} (J_m + W_m(q) + C_m) \,, \quad (3)$$

where J_m is the jitter of message m which in the worst case is equal to the worst-case response time $r_{S(m)}$ of the sender process $P_{S(m)}$, w_m is the *worst-case queuing delay* experienced by m at the communication controller, and C_m is the worst-case time it takes for a message m to reach the destination controller. On CAN, C_m depends on the frame configuration and message size s_m, while on TTP it is equal to the slot size where m is transmitted.

In Equation (3), W_m is the *worst-case queuing delay* experienced by m at the communication controller, and is calculated as:

$$W_m(q) = w_m(q) - qT_m \quad (4)$$

where q is the number of busy periods being examined, and $w_m(q)$ is the width of the level-m busy period starting at time qT_m.

The worst-case queueing delay for a message (W_m in Equation 3) is calculated differently for each type of queue:

1. The output queue of an ETC node, in which case $W_m^{N_i}$ represents the worst-case time a message m has to spend in the Out_{N_i} queue on ETC node N_i. An example of such a message is m_2 in Figure 4b, which is sent from the ETC node N_4 to the gateway node N_G, and has to wait in the Out_{N_4} queue.
2. The TTP-to-CAN queue of the gateway node, in which case W_m^{CAN} is the worst-case time a message m has to spend in the Out_{CAN} queue of node N_G. In Figure 4b, message m_1 is sent from the TTC node N_1 to the ETC node N_4, and has to wait in the

Out_{CAN} queue before it is transmitted on the CAN bus.

3. The CAN-to-TTP queue of the gateway node, where W_m^{TTP} captures the time m has to spend in the Out_{TTP} queue node N_G. Such a situation is present in Figure 4b, where message m_1 is sent from the ETC node N_4 to the TTC node N_3 through the gateway node N_G where it has to wait in the Out_{TTP} queue before it is transmitted on the TTP bus, in the S_G slot of node N_G.

On the TTC, the synchronization between processes and the TDMA bus configuration is solved through the proper synthesis of schedule tables, hence no output queues are needed. The messages sent from a TTC node to another TTC node are taken into account when determining the offsets (StaticScheduling, Figure 5), and hence are not involved directly in the ETC analysis.

The next sections show how the worst queueing delays are calculated for each of the previous three cases.

4.1.1 Worst-case queuing delays in Out_{N_i} and Out_{CAN}

The analyses for $W_m^{N_i}$ and W_m^{CAN} are similar. Once m is the highest priority message in the Out_{CAN} queue, it will be sent by the gateway's CAN controller as a regular CAN message, therefore the same equation for w_m can be used:

$$w_m(q) = B_f + \sum_{\forall f_j \in hp(f)} \left\lceil \frac{w_f(q) + J_j}{T_j} \right\rceil C_j . \quad (5)$$

The intuition is that m has to wait, in the worst case, first for the largest lower priority message that is just being transmitted (B_m) as well as for the higher priority $m_j \in hp(m)$ messages that have to be transmitted ahead of m (the second term). In the worst case, the time it takes for the largest lower priority message $m_k \in lp(m)$ to be transmitted to its destination is:

$$B_m = \max_{\forall m_k \in lp(m)} (C_k) . \quad (6)$$

Note that in our case, $lp(m)$ and $hp(m)$ also include messages produced by the gateway node, transferred from the TTC to the ETC.

4.1.2 Worst-case queuing delay in the Out_{TTP} queue

The time a message m has to spend in the Out_{TTP} queue in the worst case depends on the total size of messages queued ahead of m (Out_{TTP} is a FIFO queue), the size S_G of the gateway slot responsible for carrying the CAN messages on the TTP bus, and the frequency T_{TDMA} with which this slot S_G is circulating on the bus:

$$w_m^{TTP}(q) = B_m + \left\lfloor \frac{(q+1)s_m + I_m(w_m(q))}{S_G} \right\rfloor T_{TDMA} , \quad (7)$$

where I_m is the total size of the messages queued ahead of m. Those messages $m_j \in hp(m)$ are ahead of m, which have been sent from the ETC to the TTC, and have higher priority than m:

$$I_m(w) = \sum_{\forall m_j \in hp(m)} \left\lceil \frac{w_m + J_j}{T_j} \right\rceil s_j \quad (8)$$

where the message jitter J_m is in the worst case the response time of the sender process, $J_m = r_{S(m)}$.

The blocking term B_m is the time interval in which m cannot be transmitted because the slot S_G of the TDMA round has not arrived yet. In the worst case (i.e., the frame f has just missed the slot S_G), the frame has to wait an entire round T_{TDMA} for the slot S_G in the next TDMA round..

5. Partitioning and Mapping Strategy

At this point we can give an exact problem formulation. As an input we have an application Γ given as a set of process graphs (Section 2.1) and a two-cluster system consisting of a TT and an ET cluster. As introduced previously, $\mathcal{P}_T$ and $\mathcal{P}_E$ are the sets of processes already partitioned into the TT and ET domains, respectively. Also, $\mathcal{P}_T^M \subseteq \mathcal{P}_T$ and $\mathcal{P}_E^M \subseteq \mathcal{P}_E$ are the sets of already mapped TT and ET processes.

We are interested to find a *system configuration* denoted by a 5-tuple $\psi = <\mathcal{M}, \beta, \pi, \phi, \rho>$ such that Γ is schedulable, i.e., the imposed deadlines are satisfied. Determining a system configuration ψ means deciding on:

1. The partitioning and mapping function $\mathcal{M}$, which means deciding a partitioning for processes in $\mathcal{P}^+ = \mathcal{P} \setminus (\mathcal{P}_T \cup \mathcal{P}_E)$ and a mapping for processes in $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$, where $\mathcal{P}_T^* = \mathcal{P}_T \setminus \mathcal{P}_T^M$, and $\mathcal{P}_E^* = \mathcal{P}_E \setminus \mathcal{P}_E^M$;
2. The slot sizes and sequence β on the TTC, and the process and message priorities π on the ETC;
3. The scheduling of Γ, determining the offsets ϕ and worst-case response times ρ.

The problem, as formulated above, is NP complete and can be divided into several sub-problems: partitioning and mapping, scheduling and schedulability analysis, and bus access optimization. In our MultiClusterConfiguration strategy the design space exploration is guided by a branch-and-bound partitioning and mapping algorithm which is searching for a schedulable solution.

The MultiClusterConfiguration (MCC) strategy, presented in Figure 6, has three steps:

1. In the first step (line 2) we decide very quickly on initial values for the TTC bus access β, and the ETC priorities π. The initial TTC bus access configuration β is determined by assigning in order nodes to the slots ($S_i = N_i$) and fixing the slot length to the minimal allowed value, which is equal to the length of the largest message in the application. ET priorities π are determined using the HOPA heuristic [9], where priorities in a distributed real-time system are determined based on the local deadlines, which are calculated for each activity considering the end-to-end (global) deadlines.
2. In the second step (line 4 in Figure 6) we use a branch-and-bound algorithm to determine that partitioning and mapping $\mathcal{M}$ which leads to a schedulable solution. The BBMapping (described in Section 5.1) takes as input the application Γ, the

```
MultiClusterConfiguration(Γ)
 1 -- determining initial values for the TTC bus access and the ETC priorities
 2 <β, π> = InitialConfiguration(Γ)
 3 -- BBMapping uses MultiClusterScheduling to check schedulability
 4 <M, φ, ρ> = BBMapping(Γ, β, π)
 5 if application Γ is schedulable with the configuration <M, β, π, φ, ρ> then
 6     return schedulable
 7 else
 8     <β, π> = BusAccessOptimization(Γ, M, φ, ρ)
 9     if application Γ is schedulable with <M, β, π, φ, ρ> then
10         return schedulable
11     else
12         return not schedulable
13     end if
14end if
end MultiClusterConfiguration
```

Figure 6. The General Strategy

initial values for β and π, and produces the mapping $\mathcal{M}$. BBMapping tests the schedulability of a solution using the MultiClusterScheduling algorithm, which produces the offsets (earliest start times) ϕ and the worst-case response times ρ. MCC stops if the application is schedulable with the current configuration. If no solution has been found, the BBMapping returns the best solution (corresponding to the smallest end-to-end worst-case response time of the application).

3. Finally, the third step, namely, the bus access optimization, is performed in order to improve the chances of finding a schedulable implementation. Bus access optimization tries to find that sequence of slots and their sizes on the TTC (β), and those message priorities on the ETC (π), which are best adapted to the characteristics of the system. In this paper we do not address this issue, which has been addressed by us in [17]. If the application is not schedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

5.1 Branch-and-Bound Partitioning and Mapping

In this paper, partitioning decides whether a certain process should be assigned to TTC or ETC, and mapping means assign a process to a particular node inside a cluster. Our strategy decides the partitioning of a process implicitly, by deciding on a mapping to a particular node in a TTC or an ETC cluster.

For the partitioning and mapping we will use a branch-and-bound (BB) strategy that produces a mapping (and, implicitly, a partitioning) such that the application Γ is schedulable, if such a mapping exists. It has to be emphasized that this mapping is the best one that can be produced in the context of the TTC bus configuration β and ETC priorities π, generated in step 1 of MCC. Moreover, we define a solution as schedulable if, with the current mapping $\mathcal{M}$, and the current values for β and π, the MultiClusterScheduling algorithm returns schedulable.

In order to find a solution, the BB algorithm has to visit, in the worst case, all the alternative mappings in order to find that one which leads to a schedulable application. However, as will be shown later, in practice the number of visited solutions can be drastically reduced.

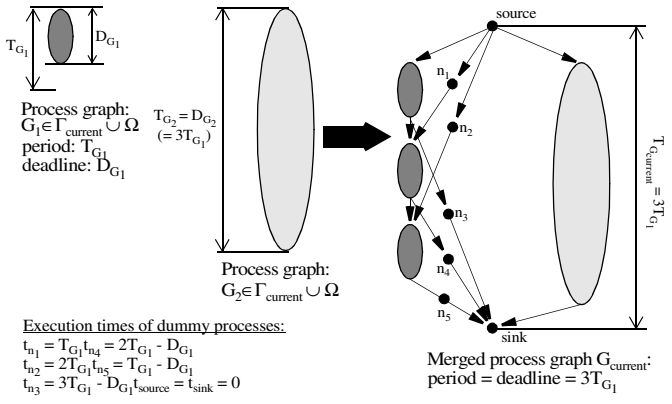

Figure 7. Graph Merging

Before applying the BB strategy we merge all the process graphs $G_i \in \Gamma$ into a single graph $\mathcal{G}$, by unrolling the process graphs and inserting dummy nodes as shown in Figure 7. The period $T_{\mathcal{G}}$ of $\mathcal{G}$ is equal to the least common multiplier of the periods T_{G_i} of the graphs G_i. Release times of processes as well as local deadlines can be easily modeled by inserting dummy nodes between certain processes and the source or the sink node, respectively. Dummy nodes (depicted with a dashed border in Figure 7) represent processes with a certain execution time but that are not to be mapped to any processor or bus. If a mapping $\mathcal{M}(P_j)$ is decided by BBMapping for a process $P_j \in G_i$, this mapping is applied to all the instances of P_j in the merged graph $\mathcal{G}$. In this context, meeting the end-to-end deadline of the merged graph $\mathcal{G}$ guarantees that each individual deadline is met.

In a branch and bound strategy, the state space corresponding to the problem is organized as a state tree. Each node σ_i corresponds to a certain state, and the children of σ_i are those states which can be reached from σ_i as result of a mapping decision. We define the partial mapping $\mathcal{M}_{\sigma_i}$ in a state σ_i as the set of all mapping decisions regarding the processes in the graph $\mathcal{G}$. Each path from the root of the tree to a leaf node corresponds to a possible solution obtained after a sequence of decisions. We are interested in finding the leaf node σ_{leaf}, such that the path from the root to σ_{leaf} corresponds to a mapping leading to a schedulable application. The schedulability of an application is determined for each leaf node σ_{leaf} by running the MultiClusterScheduling algorithm in Figure 5 with the mapping $\mathcal{M}_{\sigma_{leaf}}$.

Let us consider the example in Figure 8, where we have an application consisting of four processes, P_1 to P_4, and an architecture of three nodes, N_1 on the TTC, N_2 on the ETC, and the gateway node N_G. P_1 and P_4 are mapped on N_1, and we have to decide the mapping, and implicitly partitioning, of P_2 and P_3 to N_1 or N_2. The worst-case execution times are depicted in the table in Figure 8. The state tree corresponding to this setting is presented in Figure 8c, and has eleven states, σ_0 to σ_{10}. The leaf nodes are represented with a thicker border.

BB is based on the idea to visit only a part of the state tree without missing any state which can lead to a schedulable solution, if one exists. The main features of a BB algorithm are the *branching rule*, the *selection rule*, and the *bounding rule*. Their definition has a decisive influence on the number of visited states and, thus, on the performance of the algorithm. These rules are presented in the next three sections.

5.1.1 Branching Rule

The branching rule defines the steps which are performed for the generation of new states starting from a given parent state. An unmapped/unpartitioned process $P_i \in \mathcal{P}^*$ is considered for a mapping decision only if all its predecessors in the graph $\mathcal{G}$ have already been mapped. Let $\mathcal{L}_{\sigma_k}$ be the set of processes considered for mapping in the current state σ_k. In state σ_0 in Figure 8, $\mathcal{L}_{\sigma_0} = \{P_2, P_3\}$. We select a process $P_i \in \mathcal{L}_{\sigma_k}$ and then generate the new states containing all possible mapping decisions for P_i. That process $P_i \in \mathcal{L}_{\sigma_k}$ is selected, which has the largest *critical path* l_i, defined as:

$$l_i = \max_k \sum_{\forall \tau_j \in \pi_{ik}} r_{\tau_j} \qquad (9)$$

where π_{ik} is the k^{th} path from process P_i to the sink node of $\mathcal{G}$, and r_{τ_j} is the worst-case response time of a process or message on π_{ik}.

The worst-case response times are calculated using the MultiClusterScheduling function as follows:

$$r_{\tau_i} = \begin{cases} C_{\tau_i}^{N_k} & \text{if process } \tau_i \text{ is mapped on the TTC node } N_k \\ r\tau_i & \text{calculated using Equation (1), if } \tau_i \text{ is mapped on an ETC node} \\ C_{\tau_i}^{N_T^0} & \text{if process } \tau_i \text{ is partitioned to the TTC} \\ C_{\tau_i}^{N_E^0} & \text{if process } \tau_i \text{ is partitioned to the ETC} \\ C_{\tau_i}^{N^0} & \text{if process } \tau_i \text{ is unpartitioned} \\ r\tau_i & \text{calculated using Equation (3), if msg. } \tau_i \text{ has both sender } P_{S(\tau_i)} \text{ and destination } P_{D(\tau_i)} \text{ processes mapped} \\ w_m^{CAN} & \text{calculated using Equation (5), if message } \tau_i \text{ has } P_{S(\tau_i)} \text{ partitioned or mapped on TTC, and } P_{D(\tau_i)} \text{ partitioned or mapped on ETC} \\ w_m^{TTP} & \text{calculated using Equation (7), if message } \tau_i \text{ has } P_{S(\tau_i)} \text{ partitioned or mapped on ETC, and } P_{D(\tau_i)} \text{ partitioned or mapped on TTC} \\ 0 & \text{if message } \tau_i \text{ has one of the sender or receiver unpartitioned (or one unmapped, but both partitioned to the same cluster)} \end{cases} \quad (10)$$

where N^0 is the fastest node in $\mathcal{N}_{\tau_i}$, N_T^0 is the fastest node in $\mathcal{N}_{\tau_i} \cap \mathcal{N}_T$, and N_E^0 is the fastest node in $\mathcal{N}_{\tau_i} \cap \mathcal{N}_E$.

Starting from the state σ_k, we generate the set $\mathcal{B}_{\sigma_k}$ of new children states containing all the possible mapping decisions regarding the selected process P_i. In Figure 8, P_3 is selected from $\mathcal{L}_{\sigma_0}$ (it has a larger critical path than P_3), and the new child states of σ_0 are: σ_1, corresponding to the decision to map P_2 on N_1, and σ_2 with P_2 on N_2.

5.1.2 Selection Rule

After the children $\mathcal{B}_{\sigma_k}$ of a node σ_k have been generated, which of the children should be selected in order to continue the branching operation? The answer is given by the selection rule. A good selection rule leads quickly to leaf nodes corresponding to high quality solutions which can be used in the bounding rule.

We have implemented a selection rules to order the states in $\mathcal{B}\sigma_k$, based on the lower bound *LB* presented in the next section. In Figure 8, the selection rule *LB* would order the states in $\mathcal{B}_{\sigma_k}$ as σ_1, σ_2.

5.1.3 Bounding Rule

The most important component of a BB algorithm is the bounding rule. Before branching from a node σ_k, a decision is taken if the exploration should continue on the selected subtree, or the subtree can be cut. This decision is based on the upper bound U, which is the smallest value found so far for the worst-case response time $r_{\mathcal{G}}$ of the merged application graph $\mathcal{G}$, and the lower bound $LB\sigma_K$, which sets a lower limit on the worst-case response time $r_{\mathcal{G}}$ corresponding to any leaf node in the subtree originating from σ_k. Thus, whenever for a certain node σ_k the inequality $LB\sigma_k > U$ holds, the corresponding subtree is cut, and consequently, not investigated.

For the lower bound, α simple approach is to use the critical path [8]. For a state σ_k, the lower bound LB^{CP} is the following[1]:

$$LB_{\sigma_k}^{CP} = \forall P_i \in \overset{max}{visited}(O_i + l_i), \quad (11)$$

where O_i is the offset of P_i (its earliest possible start time) and l_i is the critical path of process P_i as defined in Equations 9 and 10. However, such a lower bound is not tight enough, i.e., it is not close enough to the optimal value of $r_{\mathcal{G}}$, hence a large number of states have to be investigated because only few branches are cut.

1. Process P_i belongs to the set of visited processes in a state σ_k, if its mapping $\mathcal{M}(P_i)$ has been decided, but no mapping has been decided yet for any of its successor processes.

Basically, LB^{CP} considers that unmapped/unpartitioned processes on the critical path execute on the fastest processors they can be mapped on. The lower bound does not consider that processes on the critical path can be delayed by other processes, which are not on the critical path. Therefore, in this paper, we propose a new, tighter lower bound *LB*, which takes into account these delays.

We use an algorithm based on list scheduling to determine the worst-case response time $r_{\mathcal{G}}^{\sigma}$ of the application graph $\mathcal{G}$ with the partial mapping $\mathcal{M}\sigma$ corresponding to a state σ. This worst-case response time value is then used as the lower bound *LB* for deciding the cutting of subtrees originating in state σ. Figure 8a presents the schedule for the optimal mapping, while Figure 8b presents the schedule for the lower bound calculation in state σ_0.

In a state σ, the lower bound is calculated in the following way. First, we determine the critical path $\mathcal{P}_{CP}$ of the application graph $\mathcal{G}$ having the partial mapping $\mathcal{M}\sigma$, as outlined in Section 5.1.1. The unpartitioned/unmapped processes P_i on the critical path are mapped on the fastest node from the list of potential nodes $\mathcal{N}_{P_i}$. With this mapping, which is a possibility in the optimal case, the critical path is the shortest. The critical path in the application graph in Figure 8 is composed of processes P_1, P_2 and P_4.

Next, the processes are scheduled using the following three rules, which guarantee that $r_{\mathcal{G}}^{\sigma}$ is smaller than the *optimal* worst-case response time $r_{\mathcal{G}}$ of the graph $\mathcal{G}$, and thus $r_{\mathcal{G}}^{\sigma}$ can be used as the lower bound *LB*.

1. The communication among the processes on the critical path is ignored.
2. Those processes for which the mapping was set[2], are scheduled

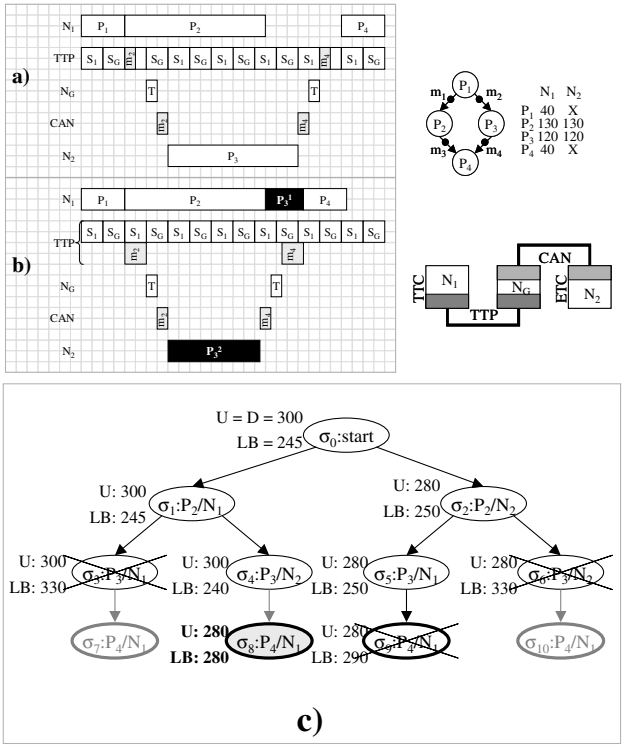

Figure 8. Branch and Bound Example

2. Mapping can be set by the designer, by previous BB decisions, or if the processes are on the critical path and thus, for the purposes of the lower bound, have been mapped on their fastest processor.

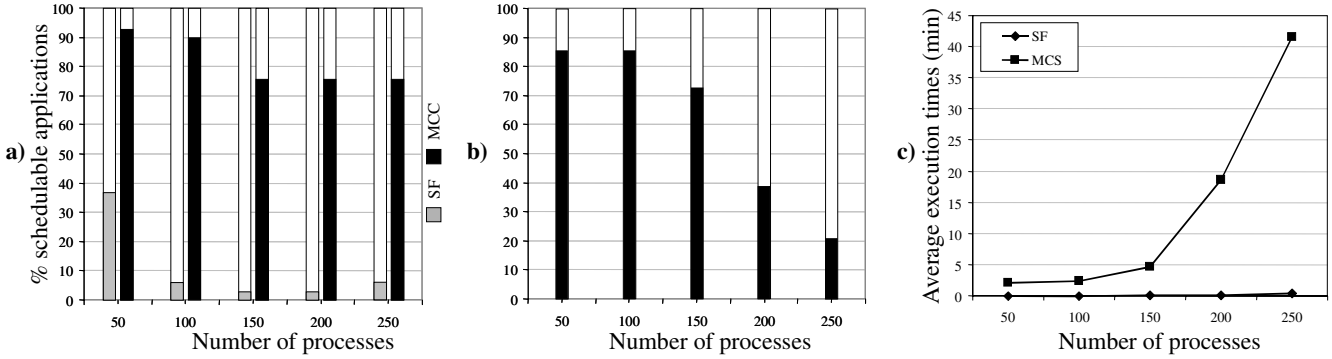

Figure 9. Evaluation of the BB algorithm

according to the MultiClusterScheduling function. The rest of the processes have to be scheduled such that the schedule length is guaranteed shorter than the optimal case. In state σ_0 of Figure 8c, for example, P_1 and P_4 are scheduled according to our multi-cluster scheduling algorithm presented in Section 4, while P_2 and P_3 are scheduled according to the next rule.

3. The load of an unpartitioned/unmapped processes can utilize the parallel resources at its disposal with ideal efficiency, subject to slack (free space in the schedule) availability and precedence constraints. This means that the load of a process P_i can be balanced simultaneously in the available slack on all its potential nodes $\mathcal{N}_{P_i}$, subject to slack and precedence constraints.

Process P_3 in Figure 8b is a process scheduled according to rule 3 in state σ_0. Its total load is 120 ms. In Figure 8b, the load of P_3, depicted as black rectangles, is distributed on two processors, N_1 and N_2. Thus, load P_3^1 is scheduled on node N_1 (35 ms), and load P_3^2 on N_2 (85 ms). However, before starting its load on N_2, P_3 would have to receive message m_2 from process P_1, as depicted in Figure 8b. The message communication delay in this case is assumed ideal, i.e., the message gets transmitted immediately on the TTC bus, regardless of the slots sequence, and it is the highest priority message on the ETC bus. Such ideal communication times are in Figure 8b those for messages m_2 and m_4 on the TTP, and m_2 and m_4 on the CAN. In addition, we do not consider the interference (see Equation (1)) among processes scheduled according to rule 3, because in the optimal case, the offsets for ET processes could be set such that the interference is eliminated.

Figure 8c presents the values for the upper bound U and the lower bound LB in each state. We are able to cut the subtrees originating from states σ_3, σ_6 and σ_9, and the optimal solution of 280 ms (its schedule depicted in Figure 8a) is found in state σ_8. When deciding if a branch should be cut, we first use LB^{CP}. If cutting is unsuccessful, we use the more tight, but also more time-consuming, lower bound LB.

6. Experimental Results

For the evaluation of our algorithms we used applications of 50, 100, 150, 200, and 250 processes (all unpartitioned and unmapped), to be implemented on two-cluster architectures consisting of 2, 4, 6, 8, and 10 different nodes, respectively, half on the TTC and the other half on the ETC, interconnected by a gateway.

Thirty examples were randomly generated for each application dimension, thus a total of 150 applications were used for experimental evaluation. We generated both graphs with random structure and graphs based on more regular structures like trees and groups of chains. Execution times and message lengths were assigned randomly using both uniform and exponential distribution within the 10 to 100 ms, and 2 to 8 bytes ranges, respectively. The experiments were done on SUN Ultra 10 computers.

We were interested to evaluate the proposed approaches. Hence, we have implemented each application, on its corresponding architecture, using the MultiClusterConfiguration (MCC) strategy from Figure 6, which uses BBMapping (BB) for the partitioning and mapping step. We have set a limit for the execution time of MCC, based on the size of the application: MCC will be stopped after four hours for 50–150 processes, eight hours for 200 processes, and 24 hours for 250 processes.

Figure 9a presents the number of schedulable solutions found after using MCC. Our MCC strategy will find a schedulable solution, if one exists. MCC will fail to find such a solution, if no mapping exists which leads to a schedulable application, or if the imposed execution time limit has been reached before finding a schedulable solution. Together with the MCC strategy, Figure 9a also presents a straightforward solution (SF). The SF approach performs a partitioning and mapping that tries to balance the utilization of resources and to minimize the communication. The SF algorithm will map a process P_i to the least utilized (in the current mapping step) processor in $\mathcal{N}_{P_i}$, as long as it does not introduce a large amount of communication. If the communication delay introduced is over a threshold, the next least utilized processor is considered for mapping. This leads to a configuration which, in principle, could be elaborated by a careful designer without the aid of optimization tools like the one proposed in the paper.

Thus, for a given graph dimension, the first bar in Figure 9a represents the number of schedulable solutions found with SF and the second bar (to the right) corresponds to MCC. Out of the total number of applications, only 11% were schedulable with the implementation produced by SF. However, using our MCC strategy, we are able to obtain schedulable applications in 82.2% of the cases. It is easy to observe that, for all application dimensions, by performing the proposed partitioning and mapping using BB, large improvements over the straightforward configuration could be

produced. Moreover, as the applications become larger, it is more difficult for SF to find schedulable solutions, while MCC performs very well. For 250 processes, for example, MCC has been able to find schedulable implementations for 76% of the applications, compared to only 6% found by SF.

Figure 9c presents the average execution times for SF and MCC. We have eliminated those cases for which no schedulable implementation has been produced. SF executes within a few seconds for large graphs. MCC takes under 45 minutes for very large process graphs of 250 processes, while for applications consisting of 150 processes it takes on average little bit less than 5 minutes.

Moreover, in order to asses the quality of our BB approach, in Figure 9b we also present the percentage of schedulable solutions found by MCC under linear execution times. Thus, we have counted the number of schedulable solutions obtained after 5 minutes for graphs of 50 processes, up to 25 minutes (growing linearly) for 250 processes. Figure 9b presents the percentage of schedulable applications obtained with MCS in this setting. MCS scales well, being able to find a large number of schedulable configurations, in a relatively short (linearly growing) amount of time.

Finally, we considered a real-life example implementing a vehicle cruise controller (CC). The process graph that models the CC has 32 processes, and is described in [14]. The CC was mapped on an architecture consisting of five nodes: Engine Control Module (ECM) and Electronic Throttle Module (ETM) on the TTC, Anti Blocking System (ABS) and Transmission Control Module (TCM) on the ETC, and the Central Electronic Module (CEM) as the gateway. We have considered a very tight deadline of 150 ms.

In this setting, the SF approach failed to produce a schedulable implementation, leading to response time of 392 ms. However, MCS has found a schedulable solution with a response time of 148 ms, taking 32 minutes of execution time.

7. Conclusions

In this paper we have presented a partitioning and mapping strategy for real-time applications distributed over multi-cluster systems, consisting of time-triggered clusters and event-triggered clusters, interconnected via gateways.

We have proposed a multi-cluster scheduling algorithm that builds a schedule table for the time-triggered activities, and determines the worst-case response times for the event-triggered activities. Based on this scheduling algorithm, we have proposed a branch-and-bound technique for the partitioning and mapping of an application functionality on the heterogeneous nodes of a multi-cluster system, such that the timing constraints of the application are guaranteed. Our branch-and-bound technique is able to efficiently explore the design space by using a tight lower bound on the end-to-end worst-case response time of the application.

Extensive experiments using synthetic applications, as well as a real-life example, show that by using our branch-and-bound approach we are able to find schedulable implementations under limited resources, achieving an efficient utilization of the system.

Acknowledgements

The authors are grateful to the industrial partners at Volvo Technology Corporation in Göteborg, Sweden, for their close involvement and thoughtful feedback during this work.

References

[1] L. Almeida, P. Pedreiras, J. A. G. Fonseca, "The FTT-CAN Protocol: Why and How", in *IEEE Trans. on Industrial Electronics*, 49(6), 2002.

[2] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, A. J. Wellings, "Fixed Priority Pre-emptive Scheduling: An Historical Perspective," in *Real-Time Systems Journal*, vol. 8, 173–198, 1995.

[3] N. Audsley, K. Tindell, A. et. al., "The End of Line for Static Cyclic Scheduling?," in *Proceedings of the Euromicro Workshop on Real-Time Systems*, 36–41, 1993.

[4] R. Bosch GmbH, *CAN Specification Version 2.0*, 1991.

[5] R. Dobrin, G. Fohler, "Implementing Off-Line Message Scheduling on Controller Area Network (CAN)," in *8th IEEE Intl. Conference on Emerging Technologies and Factory Automation*, 2001.

[6] P. Eles, A. Doboli, P. Pop, Z. Peng, "Scheduling with Bus Access Optimization for Distributed Embedded Systems," in *IEEE Transactions on VLSI Systems*, 472-491, 2000.

[7] H. Ermedahl, H. Hansson, M. Sjödin, "Response Time Guarantees in ATM Networks," in *Proc. of the Real-Time Systems Symposium*, 1997.

[8] E. B. Fernandez, B. Bussell, "Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules," in *IEEE Transactions on Computers*, 22/8, 745–751, 1973.

[9] J. J.G. Garcia, M. G. Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems," in *Workshop on Parallel and Distributed RT Systems*, 124–132, 1995.

[10] H. Kopetz, *Real-Time Systems - Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

[11] H. Lönn, J. Axelsson, "A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications," in *Euromicro Conference on Real-Time Systems*, 142–149, 1999.

[12] P. Pleinevaux, "An Improved Hard Real-Time Scheduling for the IEEE 802.5," in *Journal of Real-Time Systems*, 4(2), 1992.

[13] T. Pop, P. Eles, Z. Peng, "Schedulability Analysis for Distributed Heterogeneous Time/Event-Triggered Real-Time Systems," in *15th Euromicro Conference on Real-Time Systems*, 2003, 257-266.

[14] P. Pop, *Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems*, Linköping Studies in Science and Technology, Ph.D. Dissertation No. 833.

[15] P. Pop, P. Eles, Z. Peng, "Scheduling with Optimized Communication for Time-Triggered Embedded Systems," in *7th International Workshop on Hardware/Software Codesign*, 1999, 178–182.

[16] P. Pop, P. Eles, Z. Peng, "Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems," in *Design, Automation and Test in Europe Conference*, 2003, 184–189.

[17] P. Pop, P. Eles, Z. Peng, V. Izosimov, M. Hellring, O. Bridal, "Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications," in *Design, Automation and Test in Europe*, 2004.

[18] J. C. Palencia, M. González Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 26–37, 1998.

[19] K. Tindell, *Adding Time-Offsets to Schedulability Analysis*, Technical Report Number YCS–94–221, University of York, 1994.

[20] K. Tindell, A. Burns, A. J. Wellings, "Calculating CAN Message Response Times," in *Control Engineering Practice*, 3(8), 1163–1169, 1995.

[21] K. Tindell, J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," in *Microprocessing & Microprogramming*, Vol. 50, No. 2–3, 1994.

[22] J. Xu, D. L. Parnas, "On satisfying timing constraints in hard-real-time systems", IEEE Transactions on Software Engineering, 19(1), 1993.